

Configuration Manual

MSc Research Project
Cloud Computing Group B

AKILESH PALANIVELU

Student ID: 23109831

School of Computing
National College of Ireland

Supervisor: SHREYAS SETLUR ARUN

**National College of Ireland
Project Submission Sheet
School of Computing**



Student Name:	Akilesh Palanivelu
Student ID:	23109831
Programme:	MSc in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	03/01/2025
Project Title:	Configuration Manual
Word Count:	2000
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	AKILESH PALANIVELU
Date:	3rd January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Akilesh Palanivelu

23109831

1 Introduction

This manual details the configuration and setup process for addressing latency issues in cloud gaming. Due to technical limitations with AWS account support, the proposed Virtual GPU (VGPU) server and edge server were not feasible. As a workaround, the VGPU server concept has been replaced with a cloud EC2 instance, and the edge server has been substituted with a local setup.

2 Technologies Used

The game was developed using the following technologies:

- **HTML**
- **JavaScript**

2.1 Game Description

The game is a simple block-based game where the block can be moved using the arrow keys. The primary file is index.html, and the code remains consistent across three environments:

1. Local Setup
2. EC2 Instance with Built-in Flask Server
3. EC2 Instance with Gunicorn Server

3 Architecture Overview

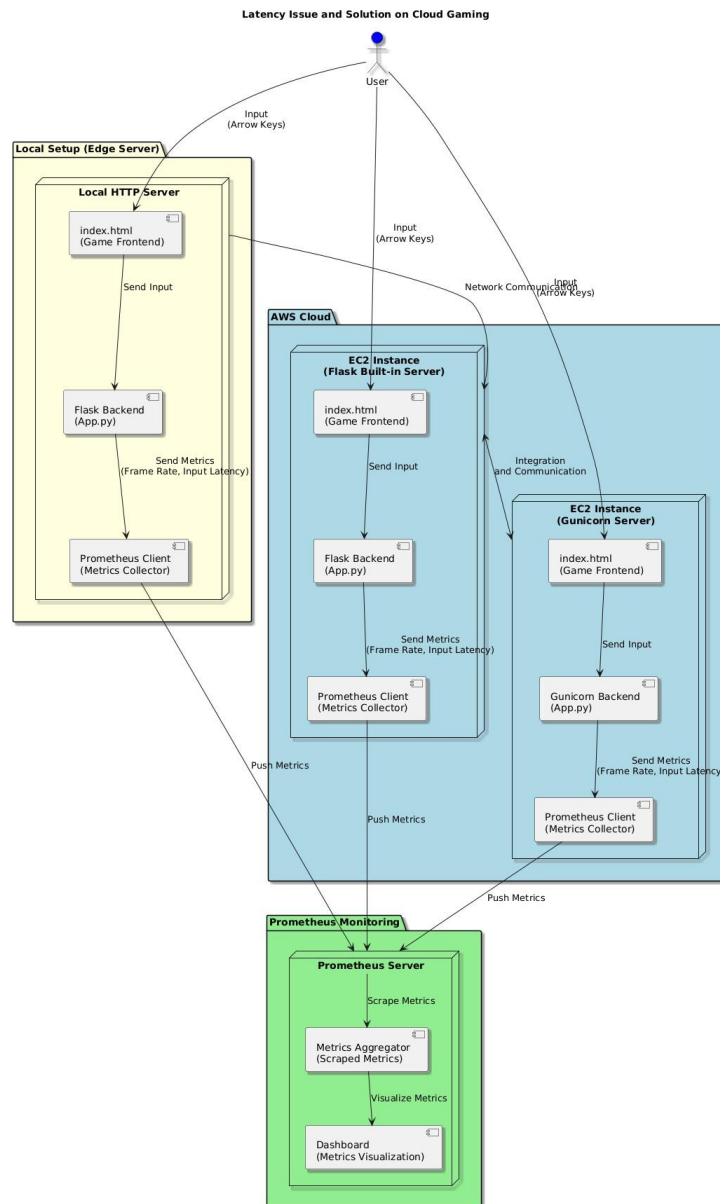
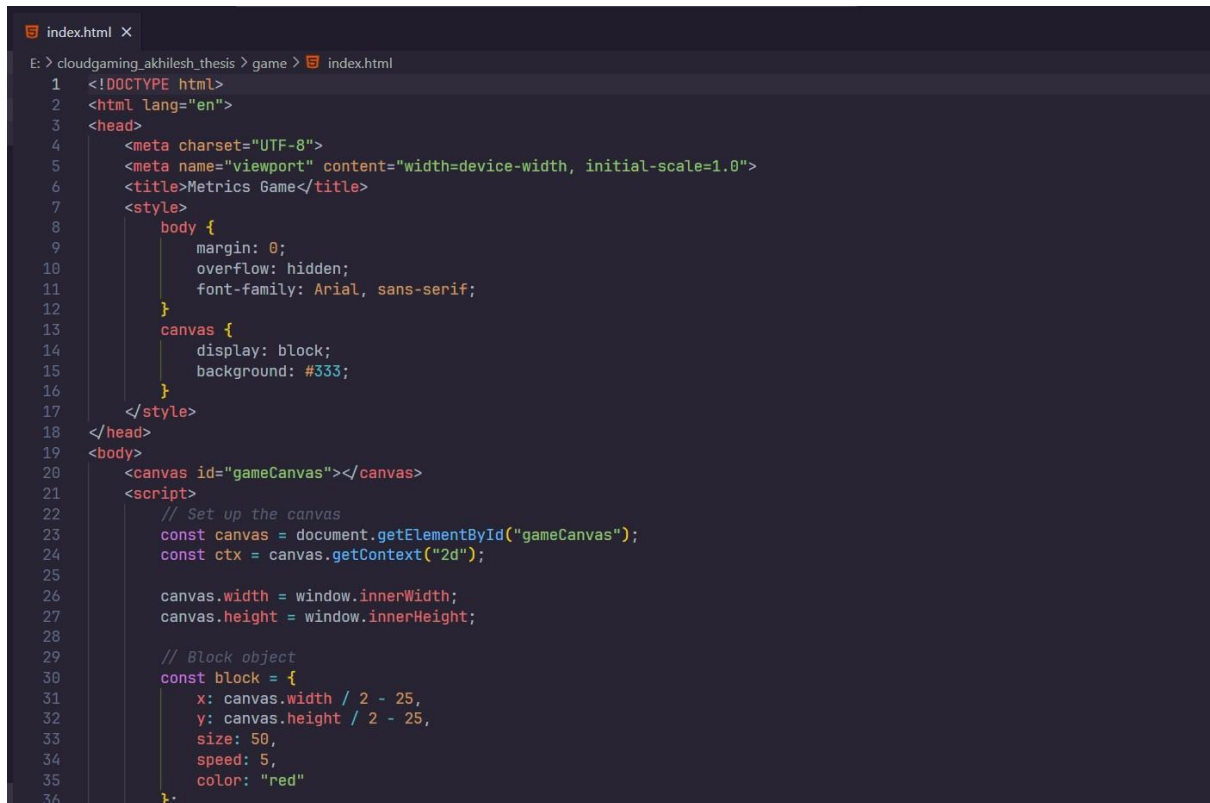


Figure 1: System Architecture Overview

Figure 1 illustrates the overall system architecture, highlighting the interactions between the local setup, EC2 instances, Flask server, Gunicorn server, and Prometheus for monitoring.

4 Game Code



```
index.html X
E: > cloudgaming_akhilesh_thesis > game > index.html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Metrics Game</title>
7      <style>
8          body {
9              margin: 0;
10             overflow: hidden;
11             font-family: Arial, sans-serif;
12         }
13         canvas {
14             display: block;
15             background: #333;
16         }
17     </style>
18 </head>
19 <body>
20     <canvas id="gameCanvas"></canvas>
21     <script>
22         // Set up the canvas
23         const canvas = document.getElementById("gameCanvas");
24         const ctx = canvas.getContext("2d");
25
26         canvas.width = window.innerWidth;
27         canvas.height = window.innerHeight;
28
29         // Block object
30         const block = {
31             x: canvas.width / 2 - 25,
32             y: canvas.height / 2 - 25,
33             size: 50,
34             speed: 5,
35             color: "red"
36         };
```

Figure 2: Frontend Code (index.html)

5 Flask Application

5.1 app.py

The Flask backend responsible for receiving and exposing metrics is detailed below:

```
index.html  app.py 1 X
E: > cloudgaming_akhilesh_thesis > app.py > ...
1 from flask import Flask, request, jsonify
2 from flask_cors import CORS
3 from prometheus_client import Gauge, generate_latest, CollectorRegistry
4
5 app = Flask(__name__)
6 CORS(app)
7
8 # Prometheus metrics
9 registry = CollectorRegistry()
10 frame_rate_gauge = Gauge('frame_rate', 'Frames per second (FPS) of the game', registry=registry)
11 input_latency_gauge = Gauge('input_latency', 'Latency of user input in milliseconds', registry=registry)
12 response_time_gauge = Gauge('response_time', 'Response time of the game in milliseconds', registry=registry)
13
14 @app.route('/metrics', methods=['POST'])
15 def receive_metrics():
16     data = request.get_json()
17     if not data:
18         print("No data received")
19         return jsonify({"status": "error", "message": "No data received"}), 400
20
21     print(f"Received Data: {data}") # Debug print statement
22     frame_rate_gauge.set(data.get("frame_rate", 0))
23     input_latency_gauge.set(data.get("input_latency", 0))
24     response_time_gauge.set(data.get("response_time", 0))
25
26     return jsonify({"status": "success", "message": "Metrics received"}), 200
27
28 @app.route('/metrics', methods=['GET'])
29 def expose_metrics():
30     # Expose metrics for Prometheus
31     return generate_latest(registry), 200, {'Content-Type': 'text/plain; charset=utf-8'}
32
33 if __name__ == "__main__":
34     app.run(host="0.0.0.0", port=5000)
35
```

Figure 3: Flask Backend Code (app.py)

6 Setup Instructions

6.1 Running the Game

Execute the following command to serve index.html in all three environments:

```
python -m http.server 8000
```

6.2 Setting Up the Flask Server

1. Install Required Packages:

```
pip install flask flask-cors prometheus-client
```

2. Run the Flask Application:

```
set FLASK_APP=app.py
flask run
```

6.3 Setting Up Prometheus

1. Download Prometheus:

<https://prometheus.io/download/>

- Download the Windows version.
- 2. **Configure Environment Variables:** - Add the Prometheus folder to your system's environment variables.
- 3. **Update** prometheus.yml: Replace the existing configuration with the following:

Listing 1: prometheus.yml

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds.
                        # Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds.
                           # Default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            # - alertmanager:9093

# Load rules once and periodically evaluate them according
# to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint
# to scrape: Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label 'job=<job name>' to
  # any timeseries scraped from this config.
  - job_name: "prometheus"
    static_configs:
      - targets: ["localhost:9090"]

# Flask app configuration
- job_name: "flask_app"
  scrape_interval: 5s # Scrape Flask app metrics every
                     # 5 seconds
  static_configs:
    - targets: ["127.0.0.1:5000"]
```

4. Run Prometheus:

```
prometheus.exe --config.file=prometheus.yml
```

5. **Verify Targets:** - Navigate to <http://127.0.0.1:9090/targets> to check the status of targets.
6. **Query Variables on Prometheus Dashboard:** - `frame_rate` - input latency
- `response_time`

7 Deployment on EC2 Instances

7.1 Local Setup

- **Game URL:** <http://127.0.0.1:8000>
- **Metrics URL:** <http://127.0.0.1:5000/metrics>
- **Prometheus URL:** <http://127.0.0.1:9090/query>

7.2 EC2 Instance with Built-in Flask Server

- **Game URL:** <http://3.110.119.8/>
- **Metrics URL:** <http://3.110.119.8:5000/metrics>
- **Prometheus URL:** <http://3.110.119.8:9090/query>

7.3 EC2 Instance with Gunicorn Server

- **Game URL:** <http://65.1.130.2:8000/>
- **Metrics URL:** <http://65.1.130.2:5000/metrics>
- **Prometheus URL:** <http://65.1.130.2:9090/query>

7.4 Setting Up Prometheus on EC2 with Gunicorn

1. Download Prometheus:

```
wget https://github.com/prometheus/prometheus/releases/download/v3.0.1/prometheus-3.0.1.linux-amd64.tar.gz
```

2. Extract the Archive:

```
tar -xvzf prometheus-3.0.1.linux-amd64.tar.gz  
cd prometheus-3.0.1.linux-amd64
```

3. Update prometheus.yml: Replace the existing configuration with the following:

Listing 2: prometheus.yml

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds.
                        # Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds.
                           # Default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            # - alertmanager:9093

# Load rules once and periodically evaluate them according
# to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint
# to scrape: Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label 'job=<job name>' to
  # any timeseries scraped from this config.
  - job_name: "prometheus"
    static_configs:
      - targets: ["localhost:9090"]

# Flask app configuration
- job_name: "flask-app"
  scrape_interval: 5s # Scrape Flask app metrics every
                     # 5 seconds
  static_configs:
    - targets: ["127.0.0.1:5000"]
```

8 Environment Details

8.1 Local Setup

- **Game URL:** <http://127.0.0.1:8000>
- **Metrics URL:** <http://127.0.0.1:5000/metrics>
- **Prometheus URL:** <http://127.0.0.1:9090/query>

8.2 EC2 Instance with Built-in Flask Server

- **Game URL:** `http://3.110.119.8/`
- **Metrics URL:** `http://3.110.119.8:5000/metrics`
- **Prometheus URL:** `http://3.110.119.8:9090/query`

8.3 EC2 Instance with Gunicorn Server

- **Game URL:** `http://65.1.130.2:8000/`
- **Metrics URL:** `http://65.1.130.2:5000/metrics`
- **Prometheus URL:** `http://65.1.130.2:9090/query`

9 Rationale for Environment Setup

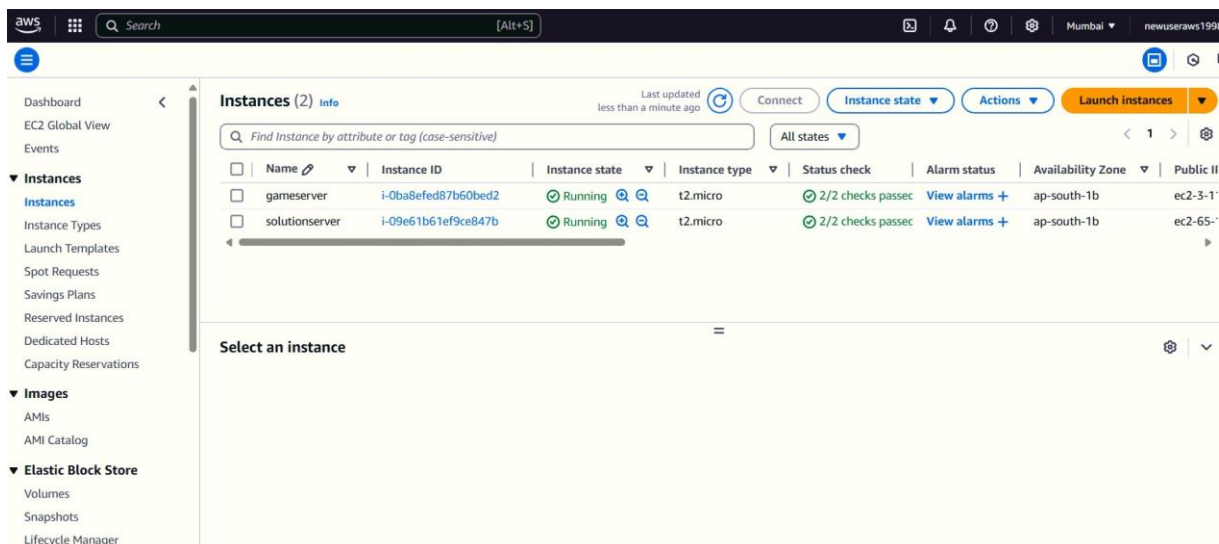


Figure 4: AWS EC2 Instances Dashboard

9.1 Purpose of Three Environments

Three distinct environments were created to simulate different server configurations and analyze their impact on game performance metrics:

1. **Cloud EC2 Instance with Built-in Flask Server:** Acts as the VGPU server.
2. **Local Setup:** Functions as the edge server.
3. **Cloud EC2 Instance with Gunicorn Server:** Integrates the edge server with the VGPU server.

9.2 Observations

Initial tests indicated that the EC2 instance with the built-in Flask server exhibited lower frame rates and higher response times compared to the local setup. This highlighted potential areas for optimization.

10 Recommendations for Reducing Latency

Based on the observations, the following suggestions are proposed to mitigate latency issues:

1. Reduce Network Latency:

- Utilize cloud regions closest to users to minimize network latency. For this project, the Mumbai server was selected.

2. Optimize Flask Application:

- Deploy a production-ready web server such as Gunicorn or uWSGI instead of relying on Flask's built-in development server.

11 Conclusion

This configuration manual outlines the setup and deployment process for addressing latency issues in cloud gaming through the use of local and cloud-based servers, Flask for backend metrics handling, and Prometheus for monitoring. By implementing the recommended optimizations, further reductions in latency and improvements in game performance metrics can be achieved.